REMARKS/ARGUMENTS

This paper responds to the Office Action of March 24, 2004.

Applicant respectfully requests reconsideration of the application. Claims 1-43 are pending, Claims 1-30 are allowed, and claim 34 is indicated allowable.

The amendments to the claims merely state cosmetic antecedents for limitations previously recited in the claims, and do not narrow their scope.

I. Claim 31

Claim 31 is discussed in paragraph 7 of the Office Action. Claim 31 reads as follows:

> 31. A method, comprising the steps of:
>
> in response to an exception raised while executing a program coded in instructions of a first instruction set architecture, initiating an execution thread under an operating system coded in instructions of a second instruction set architecture;
>
> delivering the exception to the initiated thread for handling by the operating system.

The Office Action compares the "thread" of the claim to element "500a-i" of Fig. 5b of Hammond '686. However, Hammond's elements 500a – 500i are "handlers." An "handler" is not a "thread." See the contrasting definitions from www.wordiq.com, attached as Exhibit 1. Note that the definition of "interrupt handler" makes no mention of "thread," and vice-versa – the two are essentially unrelated concepts. Similarly, the word "thread" appears nowhere in Hammond '686.

Because claim 31 recites an element that is absent from Hammond '686, any § 102 rejection may be withdrawn.

Claim 38 recites similar language and is patentable for similar reasons.

Claims 32, 33, 35-37 and 39-43 are dependent on claims 31 and 38, and patentable therewith. In addition, the dependent claims recite additional features that further distinguish the art.

II. Claims 39, 41-43

Paragraphs 14-19 of the Office Action purport to discuss certain claims under 35 U.S.C. § 103. Section 103 issues are governed by MPEP § 2143 and 2143.02. Section 2143 reads as follows:

> **2143 Basic Requirements of a *Prima Facie* Case of Obviousness**
>
> To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations.
>
> The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, not in applicant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991).

All obviousness rejections require a showing of "reasonable expectation of success." That showing is absent from paragraphs 14-19. Nor does the Office Action explain the structure or operation of any proposed combination of Hammond '686 and Thomas '563 – because they present such divergent approaches, it is not at all clear that they could be combined to yield any correct behavior. Therefore, no rejection of claims 39 or 41-43 exists.

In view of the amendments and remarks, Applicant respectfully submits that the claims are in condition for allowance. Applicant requests that the application be passed to issue in due course. The Examiner is urged to telephone Applicant's undersigned counsel at the number noted below if it will advance the prosecution of this application, or with any suggestion to resolve any condition that would impede allowance. Enclosed is Petition for Extension of Time

for one month. In the event that any further extension of time is required, Applicant petitions for that extension of time required to make this response timely. Kindly charge any additional fee, or credit any surplus, to Deposit Account No. 23-2405, Order No. 114596-28-0053BS.

Respectfully submitted,

WILLKIE FARR & GALLAGHER LLP

Dated: July 26, 2004

By: _____

David E. Boundy
Registration No. 36,461

WILLKIE FARR & GALLAGHER LLP
787 Seventh Ave.
New York, New York 10019
(212) 728-8000
(212) 728-8111 Fax

Exhibit A to

Response to Office Action

Interrupt handler

An **Interrupt Handler** is the modern progression of an interrupt service routine, a routine who's execution is triggered by a interrupt.

In modern systems Interrupt Handlers are split into two parts, the First-Level Interrupt Handler (FLIH), and the Second-Level Interrupt Handlers (SLIH).

The FLIH operates in the same way as the old *interrupt routines* did, in responce to an interrupt there is a context switch and the code for the interrupt is loaded and executed, the job of the FLIH however is not to process the interrupt, but to schedule the execution of the SLIH, while recording any critical information which is only available at the time of the interrupt.

The SLIH sits on the run queue of the operating system until it can be executed to preform the processing for the interrupt when there is processor time available.

It is worth noting that in many systems the FLIH and SLIH are referred to as *upper halves* and *lower halves*, or a derivation of those names.

Thread (computer science)

de:Thread es:hilo(informática) ja:スレッド (コンピュータプログラミング) ko:스레드 pl:Wątek (informatyka) sv:Tråd (dator) fr:Processus léger

Many programming languages, operating systems, and other software development environments support what are called "**threads**" of execution. Threads are similar to processes, in that both represent a single sequence of instructions executed in parallel with other sequences, either by time slicing or multiprocessing. Threads are a way for a program to split itself into two or more simultaneously running tasks. A common use of threads is having one thread paying attention to the graphical user interface, while others do a long calculation in the background. As a result, the application more readily responds to user's interaction.

Threads are distinguished from traditional multi-tasking processes in that processes are typically independent, carry considerable state information, and interact only through system-provided inter-process communication mechanisms. Multiple threads, on the other hand, typically share the state information of a single process, share memory and other resources directly. On operating systems that have special facilities for threads, it is typically faster for the system to context switch between different threads in the same process than to switch between different processes. Systems like Windows NT and OS/2 are said to have "cheap" threads and "expensive" processes, while in systems like Linux there is not so big a difference.

An advantage of a multi-threaded program is that it can operate faster on computer systems that have multiple CPUs, or across a cluster of machines. This is because the threads of the program naturally lend themselves for truly concurrent execution. In such a case, the programmer needs to be careful to avoid race conditions, and other non-intuitive behaviors. In order for data to be correctly manipulated, threads will often need to rendezvous in time in order to process the data in the correct order. Threads may also require atomic operations (often implemented using semaphores) in order to prevent common data from being simultaneously modified, or read while in the process of being modified. Careless use of such

primitives can lead to deadlocks.

Use of threads in programming often causes a state inconsistency. A common anti-pattern is to set a global variable, then invoke subprograms that depend on its value. This is known as accumulate and fire.

Operating systems generally implement threads in either of two ways: preemptive multithreading, or cooperative multithreading. Preemptive multithreading is generally considered the superior implementation, as it allows the operating system to determine when a context switch should occur. Cooperative multithreading, on the other hand, relies on the threads themselves to relinquish control once they are at a stopping point. This can create problems if a thread is waiting for a resource to become available. The disadvantage to preemptive multithreading is that the system may make a context switch at an inappropriate time, causing priority inversion or other bad effects which may be avoided by cooperative multithreading.

The Java programming language is an example of a computer language which supports multi-threaded computer programs.

Hardware support for software threads is provided by simultaneous multithreading. This feature was introduced in Intel's Pentium 4 processor, with the name *Hyper-threading*.

An unrelated use of the term **thread** is for threaded code, which is a form of code consisting entirely of subroutine calls, written without the subroutine call instruction, and processed by an interpreter or the CPU. Two threaded code languages are Forth and early B programming languages.

Table of contents

Programming tools

- Apache Portable Runtime (APR)
- Boost Threads
- BSP-Library
- Cheap Threads

- Common C++
- C++ Portable Types Library
- C++ Threads
- Cthreads
- Filament Runtime System
- GNU Portable Threads library (Pth)
- JThreads/C++
- Liboop
- Libthreadedserver
- Multithreaded Server Class
- netthreadedserver
- OpenThreads
- OpenTop
- pthread
- QuickThreads
- RT++
- SafePt
- TThread
- Wefts++
- Win32
- Wonderlib
- ZThreads

See also

- Thread safety
- Threading model
- green threads
- sockets
- servers
- worker
- communication
- signals
- completion port
- synchronization
- priority inversion
- Occam

- SR language
- OpenMP
- CORBA
- .NET
- DotGNU Execution Environment
- ProActive
- Adaptive Communication Environment (ACE™)
- clone()
- fork()
- Memory allocator
- Lock-free and wait-free algorithms
- Communicating sequential processes
- Message passing
- Microkernel

References

- David R. Butenhof: *Programming with POSIX Threads*, Addison-Wesley, ISBN 0-201-63392-2
- Bradford Nichols, Dick Buttlar, Jacqueline Proulx Farell: *Pthreads Programming*, O'Reilly & Associates, ISBN 1-56592-115-1
- Charles J. Northrup: *Programming with UNIX Threads*, John Wiley & Sons, ISBN 0-471-13751-0
- Mark Walmsley: *Multi-Threaded Programming in C++*, Springer, ISBN 1-85233-146-1
- Paul Hyde: *Java Thread Programming*, Sams, ISBN 0-672-31585-8
- Bill Lewis: *Threads Primer: A Guide to Multithreaded Programming*, Prentice Hall, ISBN 0-1-3443698-9
- Andrew Binstock, Richard Gerber: *Programming with Hyper-Threading Technology*, Intel, ISBN 0-9717861-4-3
- Steve Kleiman, Devang Shah, Bart Smaalders: *Programming With Threads*, SunSoft Press, ISBN 0-13-172389-8
- Pat Villani: *Advanced WIN32 Programming: Files, Threads, and Process Synchronization*, Harpercollins Publishers, ISBN 0-87930-563-0
- Jim Beveridge, Robert Wiener: *Multithreading Applications in Win32*, Addison-Wesley, ISBN 0-201-44234-5
- Thuan Q. Pham, Pankaj K. Garg: *Multithreaded Programming with Windows NT*, Prentice Hall, ISBN 0-131-20643-5
- Len Dorfman, Marc J. Neuberger: *Effective Multithreading in OS/2*, McGraw-Hill Osborne Media, ISBN 0070178410
- Alan Burns, Andy Wellings: *Concurrency in ADA*, Cambridge University Press, ISBN 0-521-62911-X
- Uresh Vahalia: *Unix Internals: the New Frontiers*, Prentice Hall, ISBN 0-13-101908-2

External links

- Ars Technica article about multithreading, etc (*http://arstechnica.com/paedia/h/hyperthreading/hyperthreading-1.html*)
- Page 1 (*http://www.ece.utexas.edu/~valvano/EE345M/view04.pdf*) & Page 2 (*http://www.ece.utexas.edu/~valvano/EE345M/view05.pdf*), a preemptive multithreaded implementation described
- Forum (*news:comp.programming.threads*)
- Answers to frequently asked questions for comp.programming.threads (*http://www.serpentine.com/~bos/threads-faq/*)
- Frequently Asked Questions (*http://lambdacs.com/cpt/FAQ.html*), Most FAQ (*http://lambdacs.com/cpt/MFAQ.html*)
- Discussion "Writing a scalable server" (*http://groups.google.com/groups?group=comp.programming.threads&threadm=580fae16.0312210310.1410bf2b%40posting.google.com*)
- The C10K problem (*http://www.kegel.com/c10k.html*)
- Business logic processing in a socket server - thread pools (*http://www.jetbyte.com/portfolio-showarticle.asp?articleId=38&catId=1&subcatId=2*)
- System support for scalable network servers (*http://www.cs.rice.edu/CS/Systems/ScalaServer/*)
- cohort scheduling (*http://citeseer.nj.nec.com/larus02using.html*)